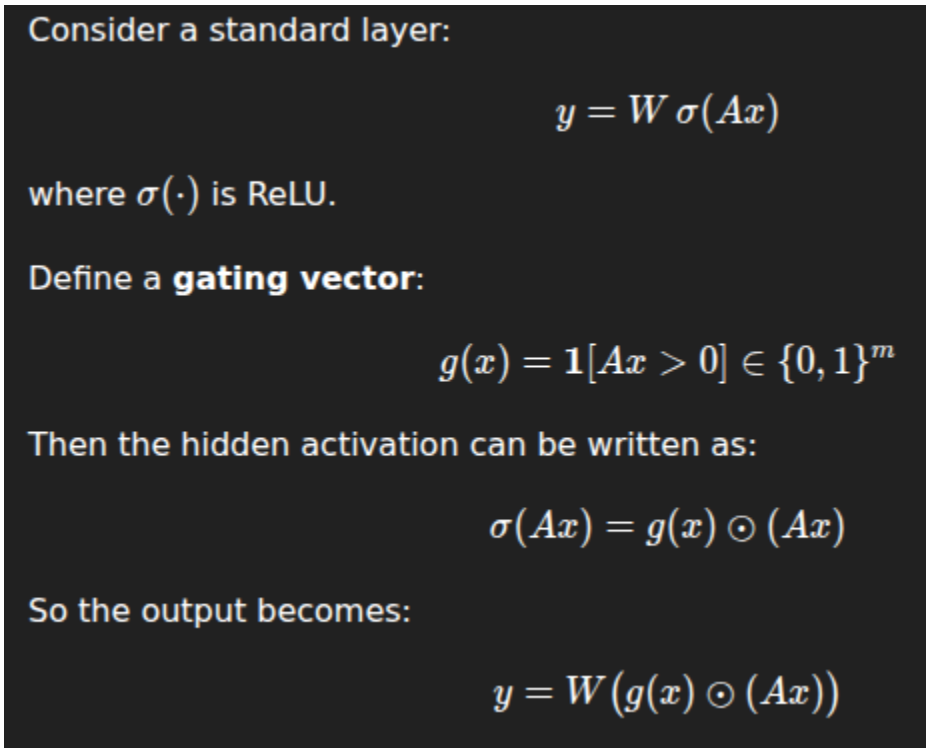
Column Selection in ReLU Networks as Hashing


Consider a standard layer:
$$y = W\,\sigma(Ax)$$
where $\sigma(\cdot)$ is ReLU.
Define a **gating vector**:
$$g(x) = \mathbf{1}[Ax > 0] \in \{0,1\}^m$$
Then the hidden activation can be written as:
$$\sigma(Ax) = g(x) \odot (Ax)$$
So the output becomes:
$$y = W\big(g(x) \odot (Ax)\big)$$

Key viewpoint

- g(x) acts as a **data-dependent binary code**
- It **selects a subset of columns** of W
- Each input activates a **subnetwork** (a column-restricted linear map)

ReLU Gating as a Hash Function

The map

$x \mapsto g(x)$

partitions input space into regions defined by hyperplanes (rows of A).

This behaves like a **hash**:

- Output is a binary vector
- Nearby inputs tend to share many bits

- Distant inputs differ in many bits

This is essentially a form of **locality-sensitive hashing (LSH)**:

- Similar inputs → similar active column sets
- Dissimilar inputs → different subnetworks

Associative Memory Interpretation

- Rewrite the output as:

$$y = \sum_{j \in \text{active}(x)} W_{:,j} \cdot (Ax)_j$$

Interpretation:

- Each column $W_{:,j}$ is a **stored feature / memory fragment**
- The gating $g(x)$ **retrieves a subset of these fragments**
- The coefficients $(Ax)_j$ weight their contribution

This matches a **distributed associative memory**:

- Address = $g(x)$ (hash / code)
- Content = columns of W
- Readout = weighted sum

Capacity and Interference

Let m be the number of columns.

Low utilization (few distinct gating patterns)

- Many columns unused per input
- System is **overdetermined**
- Effects:
    - Stable outputs
    - Some **error correction** (redundancy across columns)

High utilization (many overlapping patterns)

- Columns reused across many inputs
- Effects:
    - **Interference** between associations
    - Output noise grows (often approximately Gaussian by aggregation)

Critical regime

- When effective usage approaches capacity:
    - Weight norms increase
    - System becomes **noise sensitive**

Why Locality Matters

If gating behaves like pure random hashing:

- Small input changes flip many bits
- No generalization

ReLU improves this:

- Hyperplane partitions create **structured regions**
- Within a region, g(x) is constant → linear behavior
- Across nearby regions, only a few bits change

This gives:

- **Local generalization** (LSH-like behavior)
- Smooth degradation rather than random scrambling

Relation to Random Features

If A is random:

- g(x) approximates a **random hyperplane LSH**
- The network becomes:
    - Random embedding via (A , ReLU)
    - Linear associative readout via W

This is closely related to:

- Random feature models
- Extreme learning machines
- Sparse distributed memory systems

Key Facts to Reason With

- **Gating = hash**: $g(x) \in \{0,1\}^m$ selects active columns
- **Columns = memory atoms**: $W_{:,j}$ are reusable components
- **Output = superposition**: sum over selected columns
- **Capacity limit**: too many overlapping uses → interference
- **Noise behavior**: many small contributions → Gaussian-like error
- **Locality**: hyperplane structure ensures nearby inputs share codes

---

Takeaway

A ReLU network can be viewed as:

> A **locality-sensitive hashing scheme** that selects a subset of columns from a shared weight matrix, implementing a **distributed associative memory with linear readout**.

This framing makes it easier to reason about:

- capacity (column reuse),
- generalization (hash locality),
- and noise/interference (superposition of reused weights).